R Small Group: Class 4

Amy Pomeroy, Dayne Filer, & Sara Taylor

March 7, 2018

Using this document

- Code blocks and R code have a grey background (note, code nested in the text is not highlighted in the pdf version of this document but is a different font).
- # indicates a comment, and anything after a comment will not be evaluated in R
- The comments beginning with ## under the code in the grey code boxes are the output from the code directly above; any comments added by us will start with a single #
- While you can copy and paste code into R, you will learn faster if you type out the commands yourself.
- Read through the document after class. This is meant to be a reference, and ideally, you should be able to understand every line of code. If there is something you do not understand please email us with questions or ask in the following class (you're probably not the only one with the same question!).

Class 4 expectations

1. Implement the three basic control statements in R (for-loops, if/else statements, and while statements)
2. Use the and/or operators for combining logical statements

Control statements

Imagine you have a data set of primer sequences. You want to quickly find out information about this set of primers. Let's start by loading the list.

```
setwd("~/Documents/HLC Curriculum/HLC-Curriculum/R Materials")
primers <- read.csv("primers.csv", header = FALSE, stringsAsFactors = FALSE)
str(primers)
## 'data.frame':    560 obs. of  1 variable:
##  $ V1: chr  "GAACAAAGAAGTACAAAGGAGTAAATACAATTTTATTATCCGGATCCCCGGGTTAATTAA " "TACACCGAGTATACAACATGACT
```

There are 562 primers in the list, so if you wanted to find the length of all of them, you probably wouldn't want to count the length of each one by hand. Similarly, if you wanted to find the molecular weight of all of them, doing so by hand would be extremely time consuming. Control statements can help us make these tasks quicker.

For-loops Each programming language has its own syntax for the different control statements. In R, you create a for-loop according to the following basic syntax:

```
for (object in list) { expression with object }
```

When a for-loop is executed, R iterates through the given list and assigns each element of the list to the given variable, then executes the given expression for that variable. In the following example, the for-loop says iterate over `1:5` and use the `print` function to display each number.

```
for (a in 1:5) { print(a) }
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

If you run `ls` after running the for loop you see that R created a variable `a` and it has the value `5`.

```
ls()
## [1] "a"       "primers"
a
## [1] 5
```

The value of `a` makes sense because the last element in the given list, `1:5`, is `5`.

You can use a for-loop to do something to every element in a list. Create a list of years, called `yrs` then print a statement saying "The year is ______". You can check the length of a vector/list with the function `length`.

```
yrs <- c("2000", "2005", "2010")
for (i in 1:length(yrs)) { print(paste("The year is", yrs[i])) }
## [1] "The year is 2000"
## [1] "The year is 2005"
## [1] "The year is 2010"
```

Traditionally, for-loops iterate over the sequence `1...N` like above. However, R allows for-loops to iterate over any type of list. Rather than iterate over `1:length(yrs)` R can just iterate over `yrs`.

```
for (j in yrs) { print(paste("The year is", j)) }
## [1] "The year is 2000"
## [1] "The year is 2005"
## [1] "The year is 2010"
```

For loops can also be nested. For example, you can iterate over every element in a matrix.

```
mat <- matrix(1:6, nrow = 3)
mat
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
for (i in 1:nrow(mat)) {
  for (j in 1:ncol(mat)) {
    print(paste("mat[", i, ",", j, "] = ", mat[i, j], sep = ""))
  }
}
## [1] "mat[1,1] = 1"
## [1] "mat[1,2] = 4"
## [1] "mat[2,1] = 2"
## [1] "mat[2,2] = 5"
## [1] "mat[3,1] = 3"
## [1] "mat[3,2] = 6"
```

The example above iterates first by row, then by column. In other words, start with row 1 and go through every column, then move to the next row and iterate through every column, and so on.

Let's look at an example with our list of primers. We are interested in knowing the length of each of our primers.

```
for (i in primers[1]) {
  print(nchar(i))
}
##   [1] 61 61 62 60 60 60 60 60 60 61 61 61 60 60 60 24 24 24 24 24 24 24 24
##  [24] 30 22 24 24 26 24 24 24 27 27 24 60 60 60 60 76 60 60 60 60 60 60 60
##  [47] 21 21 21 23 24 24 26 25 22 24 30 30 54 54 30 40 47 37 20 18 25 60 60
##  [70] 60 60 60 60 60 60 60 60 60 60 22 60 60 60 27 23 35 22 22 23 20 27 60
##  [93] 60 60 60 60 60 60 60 60 20 19 20 22 23 22 22 60 29 27 21 24 48 48 22
## [116] 17 19 60 60 60 60 60 60 60 61 60 35 49 43 37 22 18 22 22 46 44 41 34
## [139] 21 18 21 61 60 61 60 60 60 60 60 60 21 22 25 23 22 24 21 23 23 23 51
## [162] 30 25 24 25 23 22 24 23 20 24 37 60 60 59 22 23 20 20 60 60 60 24 54
## [185] 61 61 18 22 60 60 60 24 60 60 60 22 60 60 60 60 60 60 60 60 60 20 22
## [208] 25 60 60 60 24 60 60 60 17 23 18 20 22 22 23 60 60 81 80 81 80 22 23
## [231] 20 20 60 60 81 80 81 80 24 20 43 40 40 19 40 40 40 22 22 60 61 20 20
## [254] 17 17 18 38 42 60 29 26 42 50 18 18 50 23 22 21 20 21 22 20 22 21 53
## [277] 60 20 71 33 33 62 18 20 51 37 45 60 35 39 55 56 35 34 35 39 49 48 47
## [300] 41 60 60 60 22 18 57 20 18 20 22 20 20 23 23 20 20 20 20 20 20 20 20
## [323] 22 23 22 20 19 22 22 60 60 22 59 60 35 34 35 35 55 52 35 30 24 22 20
## [346] 20 21 20 23 22 23 22 20 22 22 23 22 22 23 19 21 20 20 18 20 21 20 22
## [369] 21 21 21 22 19 18 19 22 21 19 21 21 60 60 21 20 60 60 22 21 60 60 22
## [392] 22 60 60 23 21 60 60 23 22 60 60 22 22 60 60 20 23 60 60 21 21 60 60
## [415] 60 22 23 60 60 23 22 60 60 22 21 60 60 22 20 60 60 21 22 60 60 23 23
## [438] 60 60 22 21 60 60 22 19 60 60 21 21 60 60 21 18 60 60 22 21 60 60 23
## [461] 22 60 60 22 22 60 60 22 22 60 60 60 22 21 60 60 22 20 60 60 60 60 22
## [484] 23 60 60 22 22 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 59 59
## [507] 59 20 22 20 23 60 60 40 40 60 60 60 60 60 60 60 60 23 22 20 20 23 24
## [530] 23 18 58 58 21 22 60 60 61 58 59 23 23 19 58 59 58 59 58 59 40 40 40
## [553] 40 40 40 40 40 40 40 27
```

The function **nchar** returns the number of characters in a string. Therefore, this code prints the number of characters in each entry in the first column of the primers data set.

This is helpful, but we might want to use this data later. To do so, we will want to assign the primer lengths to a variable rather than just printing a list of numbers.

```
len <- numeric(length = length(primers$V1))
for (i in 1:length(primers$V1)){
  len[i] <- nchar(primers[i,1])
}
```

If/else statements

In R, you create an if/else statement with the following syntax:

```
if (logical statment) { do something } else  { do something different }
```

For example:

```
if (TRUE)  { print("TRUE") } else { print("FALSE") }
## [1] "TRUE"
if (FALSE) { print("TRUE") } else { print("FALSE") }
## [1] "FALSE"
if (1 > 2) { print("TRUE") } else { print("FALSE") }
## [1] "FALSE"
```

Not all if/else statements have to have an `else` clause. By default, if no `else` statement is given R does nothing.

```
x <- 1000
if (x > 100) print("'x' is greater than 100")
## [1] "'x' is greater than 100"
```

If/else statements can also be nested:

```
y <- 25
if (y < 100) {
  if (y > 10) {
    print("10 < y < 100")
  } else {
    print("y < 10 or y > 100")
  }
} else {
  print("y < 10 or y > 100")
}
## [1] "10 < y < 100"
```

Combining if/else statments

Another way to combine if/else statements is to combine the conditional statement. There are four operators for combining logical statements: `&`, `&&`, `|`, `||`. The "and" operators (`&`/`&&`) return return `TRUE` if both statements are `TRUE`. The "or" operators (`|`/`||`) return `TRUE` if either statement is `TRUE`. The The single-character operators (`&` and `|`) return a vector.

```
1 > 0 & TRUE
## [1] TRUE
FALSE | 10 ## Remember from class 1 how 10 is coerced to TRUE
## [1] TRUE
1 > 0 & c(TRUE, FALSE, FALSE, TRUE)
## [1]  TRUE FALSE FALSE  TRUE
0 > 1 & c(TRUE, FALSE, FALSE, TRUE)
## [1] FALSE FALSE FALSE FALSE
0 > 1 | c(TRUE, FALSE, FALSE, TRUE)
## [1]  TRUE FALSE FALSE  TRUE
```

The double-character operators (`&&` and `||`) will only return a single `TRUE` or `FALSE` value.

```
1 > 0 && 10 < 100
## [1] TRUE
1 > 0 && FALSE
```

```
## [1] FALSE
1 > 0 || FALSE
## [1] TRUE
1 > 0 && c(TRUE, FALSE)
## [1] TRUE
1 > 0 && c(FALSE, TRUE)
## [1] FALSE
```

Notice in the last two statements above the order of the vector matters. The double-character operators are lazy – they only check as much as they need to. If you give a double-character operator a vector, it only checks the first element.

Furthermore, if the first expression is `TRUE` the `||` operator will not even evaluate the second expression. Similarly, if the first expression is `FALSE` the `&&` operator will not evaluate the second expression.

```
TRUE  || r
## [1] TRUE
TRUE  && r
## Error in eval(expr, envir, enclos): object 'r' not found
FALSE || r
## Error in eval(expr, envir, enclos): object 'r' not found
FALSE && r
## [1] FALSE
```

You can combine logical expressions within an if/else statement (recall we defined `y` as `25` in an earlier expression).

```
if (y < 100 && y > 10) {
  print("10 < y < 100")
} else {
  print("y < 10 or y > 100")
}
## [1] "10 < y < 100"
```

Class 4 exercises

These exercises are to help you solidify and expand on the information given above and in the supplemental material.

1. Write a for loop that prints the first 3 nucleotides of each primer. Hint: you might want to use the `substr()` function.
2. Write a for loop that prints "primer is longer than 24 bases" if the primer is longer than 20 bases, "primer is exactly 24 bases" if the primer is 20 bases, and "primer is shorter than 24 bases" if the primer is shorter than 24 bases.
3. Write a for loop that prints "primer is between 20 and 24 bases" if the primer has 20 or more bases and 24 or fewer bases.
4. Challenge. Write a script that calculates the melting temperature (Tm) of each primer. For primers that are 13 bases or less the formula for Tm in Celcius is: $Tm = (A + T) * 2 + (G + C) * 4$, where A, T, G, and C represent the number of each base. For primers that are 14 bases or more the formula for Tm in Celcius is: $Tm = 64.9 + 41 * (G + C - 16.4)/(A + T + G + C)$. Refer to the example given

in the supplemental material. Hint: the `str_count` function in the library `stringr` can be used to count the number of times a given letter appears in a string. To load the `stringr` library simply type `library(stringr)`.

Supplemental Material

While statements

The final control statement we will discuss are while statements. Although while statements are rarely used in R, they are often used in other languages and are good to understand. The while syntax in R is:

```
while (condition) {
  do something
  update condition
}
```

Try the simple example below.

```
val <- 3
while (val < 10) {
  print(val)
  val <- val + 1
}
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

If you omit the second part that updates `val` R would have printed `3` until you manually killed the process.

Combining control statements

The following example combines for loops and if/else statements. Let's revisit our primer list example. We now want to calculate the molecular weight of each of our primers. We know that the molecular weights dCTP, dATP, dGTP, and dTTP are 467.2, 491.2, 507.2, and 482.2 g/mol respectively.

```
masses <- numeric(length=length(primers$V1)) #make empty numeric vector with the same length as the pri
for (i in 1:length(primers$V1)){
  plen <- nchar(primers[i,1]) #get length of ith primer
  pmass <- 0 #set mass of primer equal to zero
  for (j in 1:plen){
    l <- substr(primers[i,1],j,j) #get jth nucleic acid of ith primer
    if (l=="C"){
      m <- 467.2
    } else if (l=="A"){
      m <- 491.2
    } else if (l=="G"){
      m <- 507.2
    } else if (l=="T"){
      m <- 482.2
    } #get mass of jth nucleic acid of ith primer
    pmass <- pmass+m #add mass of jth nucleic acid to the mass of the primer
  }
  masses[i]<-pmass #store the mass of the ith primer in the vector "masses"
}
```

You can now look at the results:

```
masses
##   [1] 29788.2 29749.2 30144.4 29288.0 29313.0 29205.0 29205.0 29192.0
##   [9] 29212.0 29770.2 29893.2 29571.2 29195.0 29327.0 29210.0 11851.8
##  [17] 11646.8 11815.8 11619.8 11643.8 11686.8 11713.8 11713.8 14712.0
##  [25] 10543.4 11655.8 11650.8 12685.2 11655.8 11788.8 11597.8 13136.4
##  [33] 13310.4 11734.8 29246.0 29135.0 29154.0 29180.0 37030.2 29305.0
##  [41] 29297.0 29004.0 29112.0 29207.0 29250.0 29112.0 10255.2 10169.2
##  [49] 10136.2 11097.6 11569.8 11704.8 12586.2 12119.0 10559.4 11548.8
##  [57] 14862.0 14487.0 26275.8 26306.8 14698.0 19492.0 22815.4 17866.4
##  [65]  9690.0  8829.6 12266.0 29250.0 29199.0 29200.0 29331.0 29339.0
##  [73] 29326.0 29270.0 29220.0 29274.0 29411.0 29149.0 29277.0 10632.4
##  [81] 29148.0 29328.0 29322.0 13146.4 11087.6 17036.0 10647.4 10661.4
##  [89] 11229.6  9777.0 13198.4 29203.0 29286.0 29206.0 29301.0 29284.0
##  [97] 29095.0 29368.0 29275.0 29081.0  9746.0  9179.8  9813.0 10681.4
## [105] 11243.6 10856.4 10657.4 29169.0 14091.8 13178.4 10306.2 11720.8
## [113] 23511.6 23227.6 10704.4  8316.4  9265.8 29134.0 29301.0 29203.0
## [121] 29242.0 29252.0 29170.0 29301.0 29713.2 29322.0 16960.0 23938.8
## [129] 20967.6 18100.4 10721.4  8820.6 10641.4 10721.4 22540.2 21338.8
## [137] 20198.2 16542.8 10239.2  8807.6 10203.2 29745.2 29321.0 29609.2
## [145] 29317.0 29313.0 29066.0 29200.0 29114.0 29116.0 10172.2 10636.4
## [153] 12208.0 11252.6 10863.4 11743.8 10150.2 11209.6 11237.6 11096.6
## [161] 24784.2 14576.0 12223.0 11624.8 12138.0 11317.6 10752.4 11686.8
## [169] 11170.6  9780.0 11766.8 17891.4 29266.0 29337.0 29084.8 10803.4
## [177] 11052.6  9714.0  9733.0 29206.0 29317.0 29259.0 11574.8 26182.8
## [185] 29824.2 29573.2  8820.6 10601.4 29259.0 29512.0 29290.0 11580.8
## [193] 29347.0 29276.0 29235.0 10625.4 29112.0 29255.0 29180.0 29286.0
## [201] 29253.0 29250.0 29303.0 29156.0 29222.0  9666.0 10853.4 12125.0
## [209] 29153.0 29137.0 29230.0 11721.8 29186.0 29206.0 29210.0  8354.4
## [217] 11101.6  8700.6  9837.0 10670.4 10672.4 11121.6 29139.0 29313.0
## [225] 39294.2 38989.0 39318.2 38964.0 10690.4 11185.6  9745.0  9806.0
## [233] 29194.0 29221.0 39319.2 38996.0 39352.2 38962.0 11590.8  9708.0
## [241] 21031.6 19477.0 19443.0  9269.8 19336.0 19584.0 19449.0 10792.4
## [249] 10641.4 29106.0 29715.2  9610.0  9686.0  8240.4  8339.4  8741.6
## [257] 18450.6 20441.4 29167.0 14240.8 12592.2 20441.4 24298.0  8788.6
## [265]  8720.6 24312.0 11043.6 10768.4 10377.2  9797.0 10377.2 10768.4
## [273]  9766.0 10768.4 10377.2 25696.6 29181.0  9699.0 34465.2 16164.6
## [281] 15957.6 30179.4  8887.6  9828.0 24726.2 18131.4 21762.0 29190.0
## [289] 17177.0 18991.8 26810.0 27234.2 17080.0 16423.8 17000.0 19062.8
## [297] 23775.8 23269.6 22907.4 19758.2 29162.0 29298.0 29202.0 10745.4
## [305]  8796.6 27719.4  9705.0  8801.6  9773.0 10703.4  9665.0  9832.0
## [313] 11206.6 11248.6  9723.0  9776.0  9893.0  9643.0  9875.0  9723.0
## [321]  9682.0  9742.0 10674.4 11194.6 10706.4  9804.0  9194.8 10754.4
## [329] 10737.4 29315.0 29202.0 10663.4 28895.8 29203.0 16999.0 16656.8
## [337] 17176.0 17077.0 26809.0 25334.4 17079.0 14509.0 11670.8 10611.4
## [345]  9828.0  9692.0 10230.2  9799.0 11296.6 10850.4 11296.6 10817.4
## [353]  9810.0 10712.4 10706.4 11163.6 10537.4 10712.4 11203.6  9351.8
## [361] 10203.2  9757.0  9752.0  8687.6  9748.0 10177.2  9659.0 10688.4
## [369] 10225.2 10261.2 10183.2 10549.4  9180.8  8709.6  9394.8 10755.4
## [377] 10285.2  9167.8 10249.2 10252.2 29153.0 29438.0 10214.2  9819.0
## [385] 29220.0 29194.0 10716.4 10209.2 29347.0 29235.0 10760.4 10645.4
## [393] 29170.0 29155.0 11268.6 10310.2 29267.0 29292.0 11236.6 10761.4
## [401] 29243.0 29263.0 10685.4 10620.4 29424.0 29242.0  9841.0 11179.6
## [409] 29046.0 29290.0 10234.2 10163.2 29126.0 29169.0 29181.0 10601.4
## [417] 11214.6 29173.0 29220.0 11319.6 10814.4 29154.0 29194.0 10746.4
## [425] 10143.2 29194.0 29220.0 10654.4  9668.0 29100.0 29354.0 10181.2
## [433] 10590.4 29066.0 29194.0 11178.6 11172.6 29212.0 29267.0 10519.4
## [441] 10221.2 29131.0 29246.0 10676.4  9238.8 29122.0 29183.0 10119.2
## [449] 10166.2 29351.0 29542.0 10203.2  8762.6 29225.0 29207.0 10816.4
## [457] 10270.2 29204.0 29140.0 11268.6 10779.4 29357.0 29175.0 10699.4
## [465] 10645.4 29385.0 29323.0 10723.4 10721.4 29192.0 29220.0 29368.0
## [473] 10700.4 10310.2 29268.0 29115.0 10709.4  9890.0 29366.0 29347.0
## [481] 29417.0 29135.0 10841.4 11277.6 29354.0 29236.0 10758.4 10771.4
## [489] 29126.0 29420.0 29140.0 29418.0 29158.0 29136.0 29304.0 29209.0
## [497] 29431.0 29165.0 29165.0 29146.0 29354.0 29171.0 29237.0 29357.0
## [505] 28769.8 28870.8 28747.8  9766.0 10850.4  9712.0 11219.6 29363.0
## [513] 29322.0 19587.0 19296.0 29235.0 29208.0 29265.0 29236.0 29337.0
## [521] 29123.0 29154.0 29383.0 11366.6 10665.4  9726.0  9721.0 11101.6
## [529] 11652.8 11259.6  8851.6 28102.6 28342.6 10254.2 10732.4 29026.0
## [537] 29307.0 29871.2 28317.6 28676.8 11141.6 11286.6  9233.8 28205.6
## [545] 28721.8 28108.6 28709.8 28408.6 28684.8 19640.0 19390.0 19420.0
## [553] 19440.0 19627.0 19481.0 19471.0 19502.0 19460.0 19431.0 13198.4
```